计算机视觉课程实验报告

实验二：基于卷积神经网络的 MNIST 手写体数字识别

姓　　名：　　李嘉鹏

学　　院：　　计算机科学与技术学院

专　　业：　　数据科学与大数据技术

班　　级：　　大数据 2101 班

学　　号：　　U202115652

指导教师：　　刘康

2023 年 12 月 20 日

目　录

实验二 基于卷积神经网络的 MNIST 手写体数字识别

1.1 Introduction（实验简介）

1.1.1 Background（实验背景）

卷积神经网络也是计算机视觉领域中常见的一种网络结构，它属于前馈神经网络的一种。卷积神经网络有三个结构上的特性：局部连接、权重共享、空间或时间上的次采样。卷积神经网络的基本结构如图 1 所示。

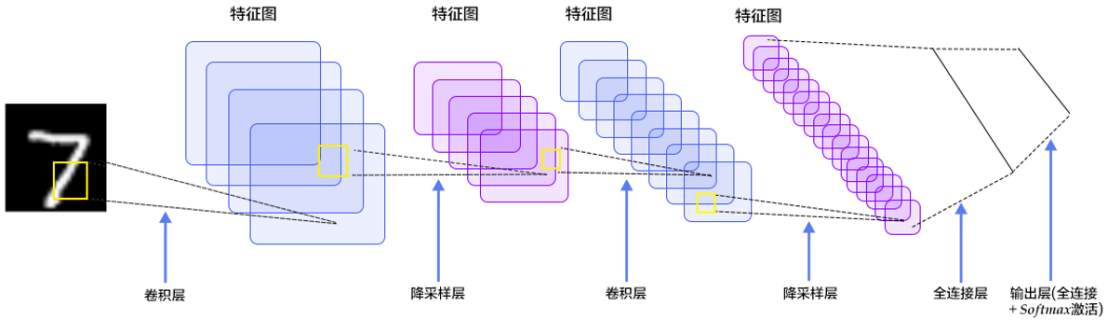

图 1：卷积神经网络结构

整个卷积神经网络主要分为卷积层（进行图像的特征提取）、池化层（减少计算量并实现平移、旋转、尺度不变性）、全连接层（将网络提取到的特征进行整合）。经典的卷积神经网络包括 ResNet、LeNet、AlexNet 和 VGG 等。其中，残差网络（Residual Network，ResNet）通过给非线性的卷积层增加直连边来提高信息的传播效率。

1.1.2 ResNet 介绍

ResNet 是一种较新的神经网络结构，它引入了残差单元来解决退化问题。其瓶颈残差模块一般依次由 1×1、3×3、1×1 这三个卷积层堆积而成，其中 1×1 卷积能起到降维或升维作用，从而使 3×3 的卷积在相对较低维度的输入上进行，提高计算效率。残差模块的构成如图 2 所示。

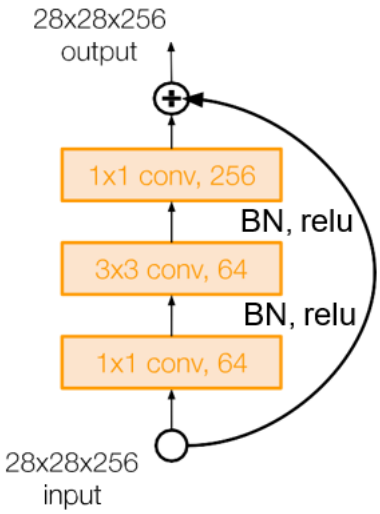

图 2：ResNet 残差模块构成示意图

该残差模块有助于解决梯度消失和退化问题，提高模型训练的效果。

1.1.3 Motivation（实验目的）

本实验是基于卷积神经网络的分类任务，要求设计一个卷积神经网络，并在其中使用 ResNet 模块，在 MNIST 数据集上实现 10 分类手写体数字识别。最终不仅需要给出模型的总体准确率，还需要给出模型在 10 个分类中每一类的准确率。

实验中可以任选深度学习框架，尝试不同的网络架构、不同的神经元个数、使用不同的激活函数和训练超参数等，观察并比较网络性能。

1.2 Proposed Method（实验设计与步骤）

1.2.1 Network Architecture of ResNet（ResNet 模块架构）

我采用的深度学习框架为 Tensorflow。

由于实验不允许直接导入现有的 ResNet 网络，因此我首先定义了 ResNet 模块（ResNetBlock 类），在构造函数中定义了三个卷积层和三个批量正则化层，分别为 conv1、bn1、conv2、bn2、conv3 和 bn3。在 call 方法中，定义了层的前向传播，通过依次调用这些层，将输入数据进行卷积、批量正则化和残差连接的操作，最后返回输出，代码如下所示。

```
class ResNetBlock(layers.Layer):
    def __init__(self, filters):
        super(ResNetBlock, self).__init__()
        self.conv1 = layers.Conv2D(filters, (1, 1), activation='relu')
        self.bn1 = layers.BatchNormalization()
        self.conv2 = layers.Conv2D(filters, (3, 3), activation='relu',
padding='same')
        self.bn2 = layers.BatchNormalization()
        self.conv3 = layers.Conv2D(filters, (1, 1))
        self.bn3 = layers.BatchNormalization()

    def call(self, inputs):
        x = self.conv1(inputs)
        x = self.bn1(x)
        x = self.conv2(x)
        x = self.bn2(x)
        x = self.conv3(x)
        x = self.bn3(x)
        x = layers.Add()([inputs, x])
        x = layers.Activation('relu')(x)
        return x
```

ResNet 模块内部的主要架构是：

（1）Conv2D 卷积层

卷积层 conv1、conv2 和 conv3 通过卷积操作提取输入数据的特征。其中，conv1 和 conv3 使用 1x1 的卷积核进行卷积操作，而 conv2 使用 3x3 的卷积核进行卷积操作，并且通过在外围 padding 的方式确保输出特征图和输入特征图的大小相同。三个卷积层的步长均为 1。这三层共同构成 ResNet 的残差模块。

（2）BatchNormalization 批量正则化层

批量正则化层 bn1、bn2 和 bn3 对卷积层的输出进行统一的批量正则化操作，加速训练过程，提高在测试集上的泛化能力。其本质就是在每一层输入时插入了一个归一化处理（归一化后数据的均值 $\overline{\mu}=0$，方差 $\sigma^2=1$），然后再进入网络的下一层。

（3）Add 层和 Activation 层

在 ResNet 模块的最后，将输入数据与经过卷积和批量正则化处理后的输出进行残差连接（add），将之前的输入直接与当前层的输出相加，从而跳过一部分网络层，把前面的特征传递给后面的层。最后，通过使用 relu 激活函数来确保输出是非线性的（激活函数可替换为其它的，例如 sigmoid 等）。需要注意的是，这里把激活函数放在最后一步，是因为如果最后一次激活函数放在第三个卷积层之后，那么残差连接之后的特征将直接受到激活函数的影响，可能导致残差信息被改变或丢失。

1.2.2 Network Architecture of CNN（卷积神经网络架构）

定义好 ResNet 模块后，现在可以直接使用封装好的 ResNet 模块实现一个完整的神经网络架构，在代码实现中我定义了一个名为 CNNModel 的类。

在 CNNModel 的构造函数中，我首先定义了卷积层 conv1、批量正则化层 bn1、最大池化层 maxpool 以及两个 ResNet 模块 resnet_block1 和 resnet_block2。在 call 方法中，我按顺序调用这些层，对输入数据进行卷积、批量正则化和残差连接操作，最后通过扁平化层 flatten 和全连接层 fc1 得到特征向量，再通过 Dropout 层 dropout 进行随机失活，最终经过输出层 fc2 得到结果，即预测出的概率分布。具体代码如下所示。

```
class CNNModel(tf.keras.Model):
    def __init__(self):
        super(CNNModel, self).__init__()
        self.conv1 = layers.Conv2D(32, (3, 3), activation='relu',
input_shape=(28, 28, 1))  # 卷积层
        self.bn1 = layers.BatchNormalization()  # 批量正则层
        self.maxpool = layers.MaxPooling2D((2, 2))  # 最大池化层
        self.resnet_block1 = ResNetBlock(32)  # ResNet 模块 1
        self.resnet_block2 = ResNetBlock(32)  # ResNet 模块 2
        self.flatten = layers.Flatten()  # 扁平化层
        self.fc1 = layers.Dense(64, activation='relu')  # 全连接层
        self.dropout = layers.Dropout(0.5) # Dropout 层
        self.fc2 = layers.Dense(10, activation='softmax') # 全连接层（也
```

```
就是输出层）

    def call(self, inputs):
        x = self.conv1(inputs)
        x = self.bn1(x)
        x = self.maxpool(x)
        x = self.resnet_block1(x)
        x = self.resnet_block2(x)
        x = self.flatten(x)
        x = self.fc1(x)
        x = self.dropout(x)
        output = self.fc2(x)
        return output
```

神经网络架构中各部分的特点和作用分别是：

（1）输入层

输入的图像大小为 28x28 像素，通道数为 1（因为 MNIST 数据集为单色图像，每个像素只有一个通道即灰度值）。

（2）卷积层

包含 32 个大小为 3x3 的卷积核，负责提取图像的特征。

（3）批量正则化层

对卷积层的输出进行批量正则化，加速收敛过程。

（4）最大池化层

进行 2x2 的最大池化操作，对特征图进行下采样，在减少数据量的同时保留关键信息。

（5）ResNet 模块

每个 ResNet 模块（ResNetBlock）由三个卷积层和一个残差连接组成，具体架构见上面的定义。特征图经过 ResNet 后输出的图像大小仍为 28x28。

（6）扁平化层

将前面输出的特征图展平为一维向量，为全连接层做准备。

（7）全连接层

包含 64 个神经元，主要负责将网络提取到的特征整合到一起。

（8）Dropout 层

Dropout 正则化，随机丢弃一部分神经元，用于避免过拟合的问题。其基本步骤如图 3 所示，主要是不断重复以下两步：①随机删掉网络中一定比例的隐藏神经元，输入输出神经元保持不变；②把输入 x 通过修改后的网络前向传播，然后对修改后的神经元进行参数更新。

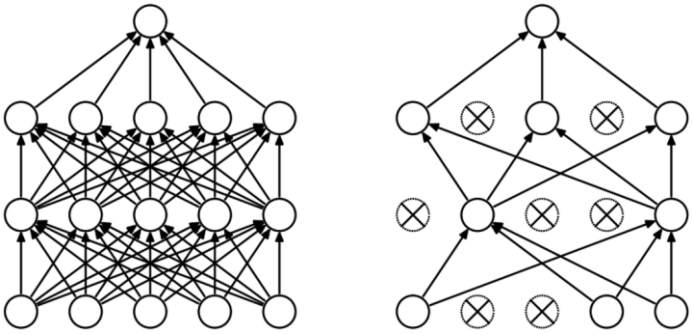

图 3：Dropout 正则化步骤示意图

（9）输出层（同时也是全连接层）

输出层，包含 10 个神经元（对应 0-9 这 10 个数字）。由于这是一个多分类问题，因此采用 softmax 激活函数用于输出预测结果的概率分布，它能更好地拟合非线性数据。最终，概率最大的神经元对应的数字将被作为预测结果输出。

综合上面的分析，一种可行的神经网络架构如图 4 所示，各层的参数在左侧列出。网络架构图使用 NN-SVG[4]绘制。

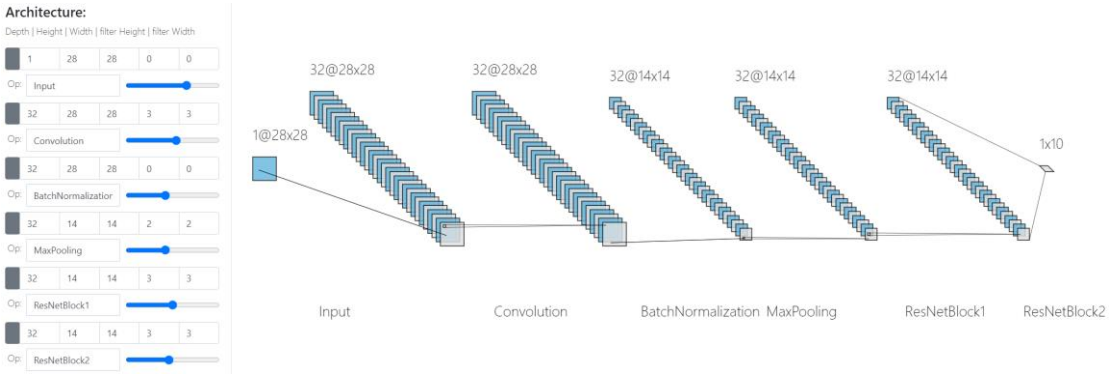

图 4：一种使用 ResNet 模块的卷积神经网络架构示意图

由于分类任务中最常用的损失函数为交叉熵损失函数，因此本实验中我使用了 sparse_categorical_crossentropy 作为损失函数来适应多分类问题。同时，我还使用了 adam 作为优化器。训练模型时每个 epoch 包含多个 mini-batch，每个 mini-batch 中包含的训练样本数量为 batch_size。epoch 和 batch_size 的取值同样也可以修改，最终使用测试集来验证模型。代码如下所示。

```
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

history = model.fit(x_train, y_train, epochs=10, batch_size=512,
validation_data=(x_test, y_test), callbacks=[LossHistory((x_train,
y_train), (x_test, y_test))])
```

对于数据集的划分，我直接使用了 MNIST 数据集自带的数据划分模式，也就是 60000 个训练集图片、10000 个测试集图片。

```
mnist = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

最后，在检验模型效果时，为了输出模型在 10 个分类中每一类的准确率，我首先获取了模型对所有 case 的预测概率，并找出概率最大的数字作为预测结果。然后，我使用了 sklearn 库中的 classification_report 函数计算每个类别的评估指标，可以输出预测精确率（precision）、**召回率（recall，即准确率）**、F1 score 和各类别的样本量（support）。代码如下所示。

```
from sklearn.metrics import classification_report

y_pred_probabilities = model.predict(x_test)
y_pred = np.argmax(y_pred_probabilities, axis=-1)

# 使用 classification_report 函数计算每个类别的评估指标
report = classification_report(y_test, y_pred, digits=10,
output_dict=False)
print(report)
```

1.2.2 Argument Selection（参数选择）

为了对比神经网络在不同的网络架构、神经元个数、激活函数和训练超参数下的性能表现，我引入了 3 组不同的网络架构参数，如表 1 所示。

表 1：神经网络架构参数选择

组	网络层数	神经元个数	激活函数	卷积核大小	epoch	batch_size
1	卷积层	32	softmax	3x3	10	512
	最大池化层	-	-	2x2		
	ResNet（模块 1） ResNet（模块 2）	ResNet 模块结构：①3 个卷积层各有 32 个卷积核，卷积核大小分别为 1x1、3x3、1x1；②步长均为 1；③输入和输出特征图大小相同；④三个激活函数均为 relu				
	全连接层 1	64	relu	-		
	Dropout	随机丢弃占比为 0.5（50%）的神经元				
	输出层	10	softmax	-		
2	卷积层	64	softmax	3x3	10	512
	最大池化层	-	-	2x2		
	ResNet（模块 1） ResNet（模块 2）	ResNet 模块结构：①3 个卷积层各有 64 个卷积核，卷积核大小分别为 1x1、3x3、1x1；②步长均为 1；③输入和输出特征图大小相同；④三个激活函数均为 relu				
	全连接层 1	64	relu	-		
	Dropout	随机丢弃占比为 0.5（50%）的神经元				
	输出层	10	softmax	-		
3	卷积层	32	softmax	3x3	10	512
	最大池化层	-	-	2x2		
	ResNet（模块 1） ResNet（模块 2） ResNet（模块 3）	ResNet 模块结构：①3 个卷积层各有 32 个卷积核，卷积核大小分别为 1x1、3x3、1x1；②步长均为 1；③输入和输出特征图大小相同；④三个激活函数均为 relu				
	全连接层 1	64	relu	-		
	Dropout	随机丢弃占比为 0.5（50%）的神经元				
	输出层	10	softmax	-		

1.3 Experimental Results（实验结果）

为了保存每一轮 mini-batch 训练后的模型在训练集和测试集上的损失(loss)，使用下面的语句运行 main.py，并将训练过程中程序输出到终端的结果保存在 output.log 文件中。

```
python main.py > output.log
```

据此，可以直接看出每次 mini-batch 训练后模型在训练集和测试集上的损失（Train loss 和 Test loss），以及其在训练集上的准确率 accuracy，如图 5 所示。由于日志文件很长，因此后续会将 loss 的变化趋势绘制为图片的形式进行展示。

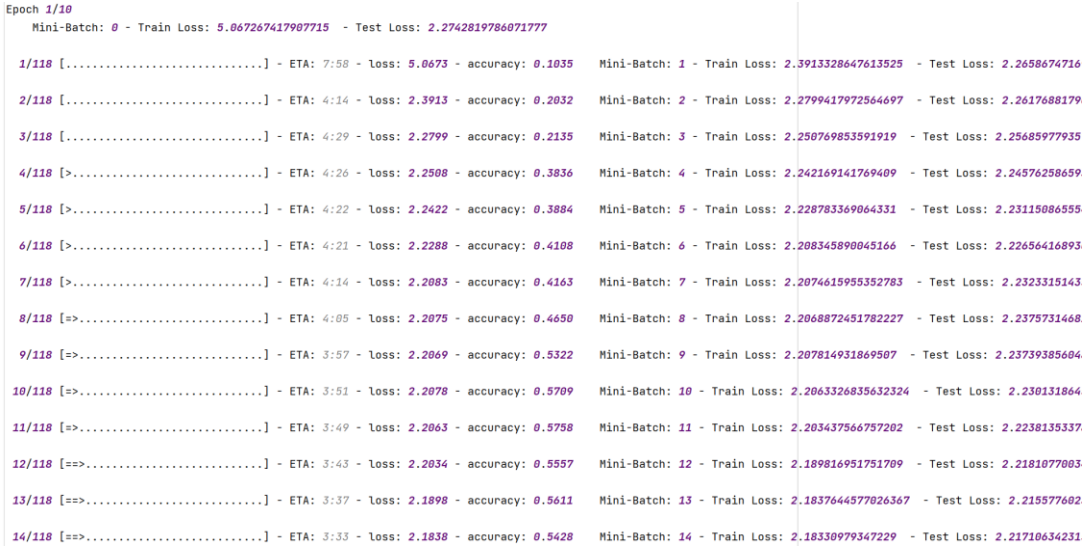

图 5：output.log 记录了每个 mini-batch 训练后模型在训练集和测试集上的损失 loss

1.3.1 第一组神经网络架构实验结果

在第一组的条件下，训练完成后模型在训练集和测试集上的损失 loss 和准确率 acc 分别如图 6 所示。此时测试集的 accuracy 达到了 0.9861。

整个训练过程中，模型在训练集和测试集上的损失 loss 随 mini-batch 数量的变化如图 7 所示。

```
Train loss: 0.045580153234350966
Train accuracy: 0.9898223001032505
Test loss: 0.058958906680345535
Test accuracy: 0.9861000180244446
```

图 6：实验结果（第一组）

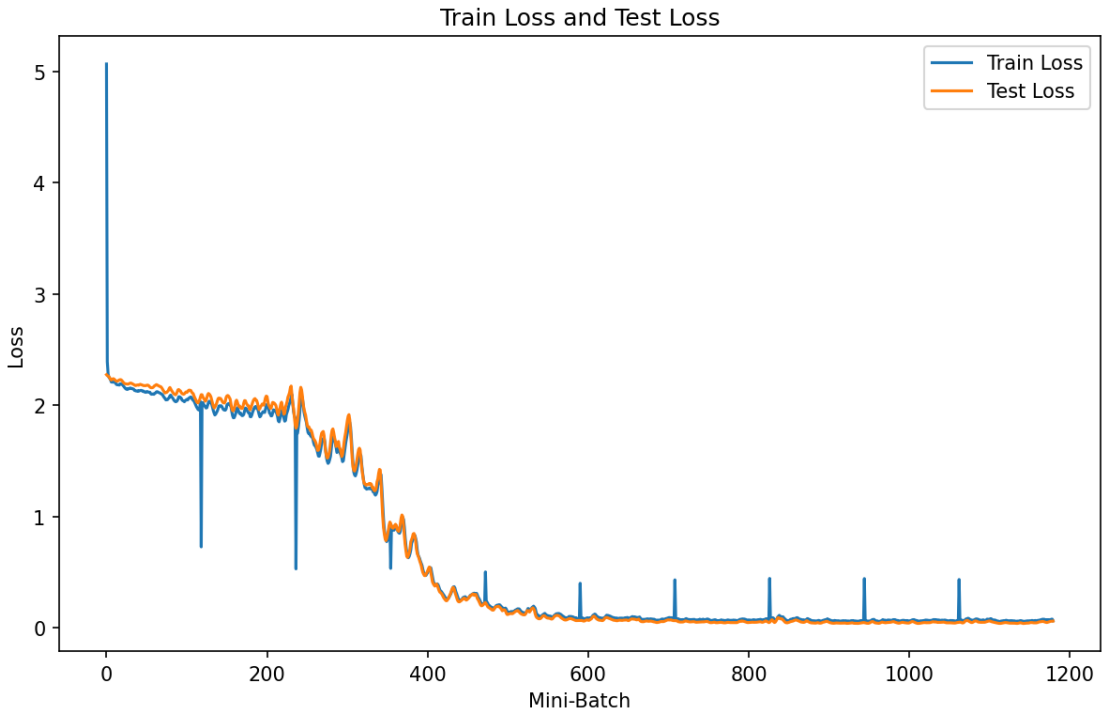

图 7：训练集和测试集上的损失 loss 随 mini-batch 数量的变化（第一组）

最终，模型在测试集十分类中每一类的准确率 recall 如图 8 所示。图中最左侧第一列为标签，第二列为精确率（precision），**第三列为召回率（recall，即所要求的准确率，代表模型识别出属于该类别的数据量与该类别在真实样本中的数量之比）**，后两列是 F1 score 和各标签对应的数据量（support）。

	precision	recall	f1-score	support
0	0.9958890031	0.9887755102	0.9923195084	980
1	0.9982253771	0.9911894273	0.9946949602	1135
2	0.9698397738	0.9970930233	0.9832775920	1032
3	0.9719806763	0.9960396040	0.9838630807	1010
4	0.9897645855	0.9847250509	0.9872383869	982
5	0.9909706546	0.9843049327	0.9876265467	892
6	0.9916054565	0.9864300626	0.9890109890	958
7	0.9940000000	0.9669260700	0.9802761341	1028
8	0.9718026183	0.9907597536	0.9811896289	974
9	0.9879396985	0.9742319128	0.9810379242	1009
accuracy			0.9861000000	10000
macro avg	0.9862017844	0.9860475347	0.9860534751	10000
weighted avg	0.9862583736	0.9861000000	0.9861069417	10000

图 8：测试集十分类中每一类的准确率（第一组）

1.3.2 第二组神经网络架构实验结果

在第二组的条件下，训练完成后模型在训练集和测试集上的损失 loss 和准确率 acc 分别如图 9 所示。此时测试集的 accuracy 达到了 0.9832。

整个训练过程中，模型在训练集和测试集上的损失 loss 随 mini-batch 数量的变化如图 10 所示。

```
Train loss: 0.09645247459411621
Train accuracy: 0.9875666499137878
Test loss: 0.11214318871498108
Test accuracy: 0.983299970626831
```

图 9：实验结果（第二组）

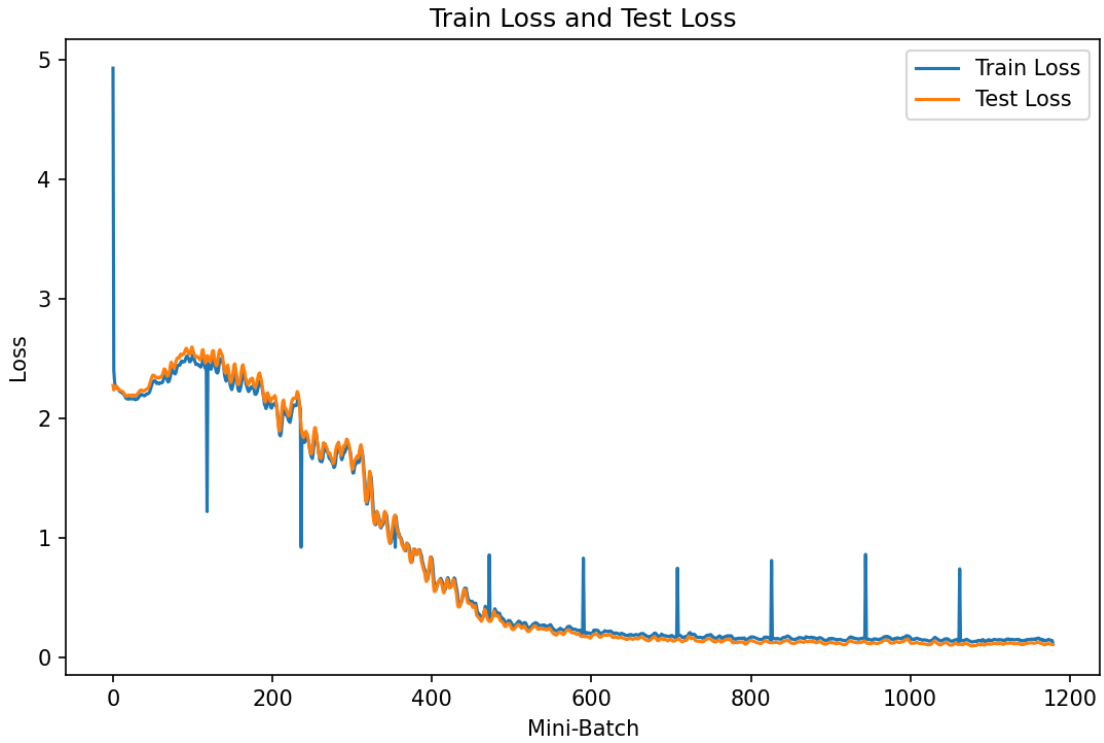

图 10：训练集和测试集上的损失 loss 随 mini-batch 数量的变化（第二组）

最终，模型在测试集十分类中每一类的准确率 recall 如图 11 所示。

	precision	recall	f1-score	support
0	0.9858585859	0.9959183673	0.9908629442	980
1	0.9877300613	0.9929515419	0.9903339192	1135
2	0.9588014981	0.9922480620	0.9752380952	1032
3	0.9861111111	0.9841584158	0.9851337958	1010
4	0.9845679012	0.9745417515	0.9795291709	982
5	0.9887640449	0.9865470852	0.9876543210	892
6	0.9936507937	0.9801670146	0.9868628481	958
7	0.9881889764	0.9766536965	0.9823874755	1028
8	0.9916492693	0.9753593429	0.9834368530	974
9	0.9703557312	0.9732408325	0.9717961405	1009
accuracy			0.9833000000	10000
macro avg	0.9835677973	0.9831786110	0.9833235564	10000
weighted avg	0.9834224658	0.9833000000	0.9833109566	10000

图 11：测试集十分类中每一类的准确率（第二组）

1.3.3 第三组神经网络架构实验结果

在第三组的条件下，训练完成后模型在训练集和测试集上的损失 loss 和准确率 acc 分别如图 12 所示。此时测试集的 accuracy 达到了 0.9888。

整个训练过程中，模型在训练集和测试集上的损失 loss 随 mini-batch 数量的变化如图 13 所示。

```
Train loss: 0.01874365843832493
Train accuracy: 0.9937833547592163
Test loss: 0.038787420839071274
Test accuracy: 0.9887999892234802
```

图 12：实验结果（第三组）

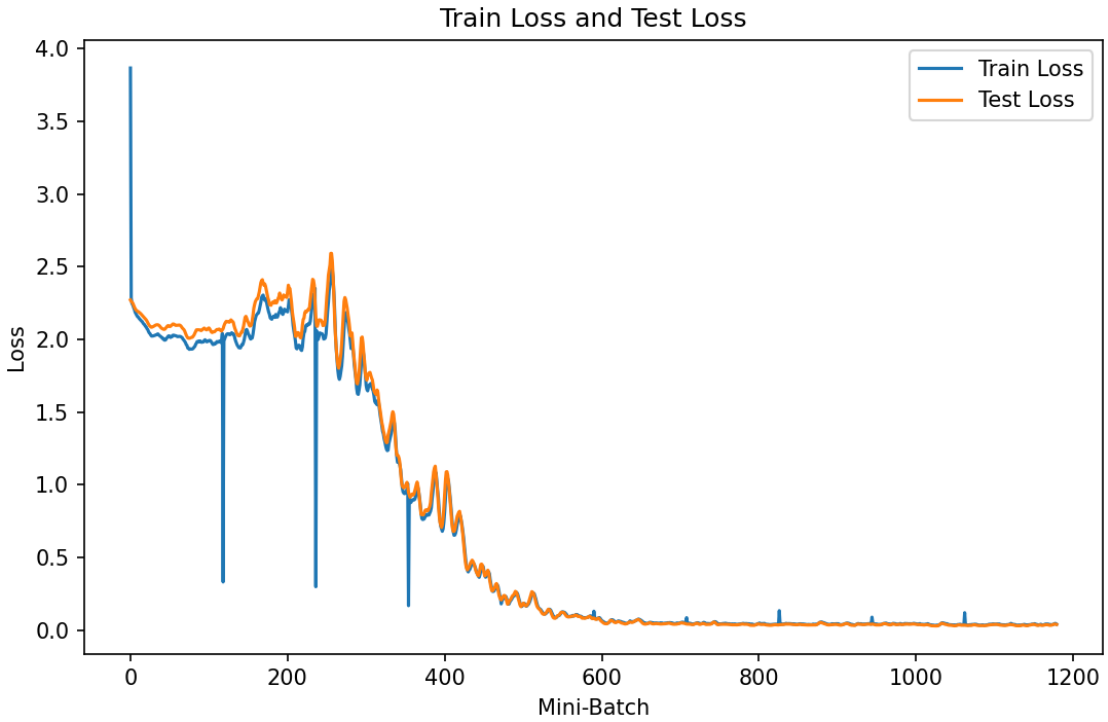

图 13：训练集和测试集上的损失 loss 随 mini-batch 数量的变化（第三组）

最终，模型在测试集十分类中每一类的准确率 recall 如图 14 所示。

	precision	recall	f1-score	support
0	0.9779559118	0.9959183673	0.9868554095	980
1	0.9929824561	0.9973568282	0.9951648352	1135
2	0.9941634241	0.9903100775	0.9922330097	1032
3	0.9960159363	0.9900990099	0.9930486594	1010
4	0.9721669980	0.9959266802	0.9839034205	982
5	0.9910414334	0.9921524664	0.9915966387	892
6	0.9895615866	0.9895615866	0.9895615866	958
7	0.9854932302	0.9912451362	0.9883608147	1028
8	0.9968553459	0.9763860370	0.9865145228	974
9	0.9918781726	0.9682854311	0.9799398195	1009
accuracy			0.9888000000	10000
macro avg	0.9888114495	0.9887241620	0.9887178717	10000
weighted avg	0.9888890805	0.9888000000	0.9887950702	10000

图 14：测试集十分类中每一类的准确率（第三组）

三组条件下，训练得到的模型在测试集十分类中每一类的准确率 recall 分布情况如图 15 所示。

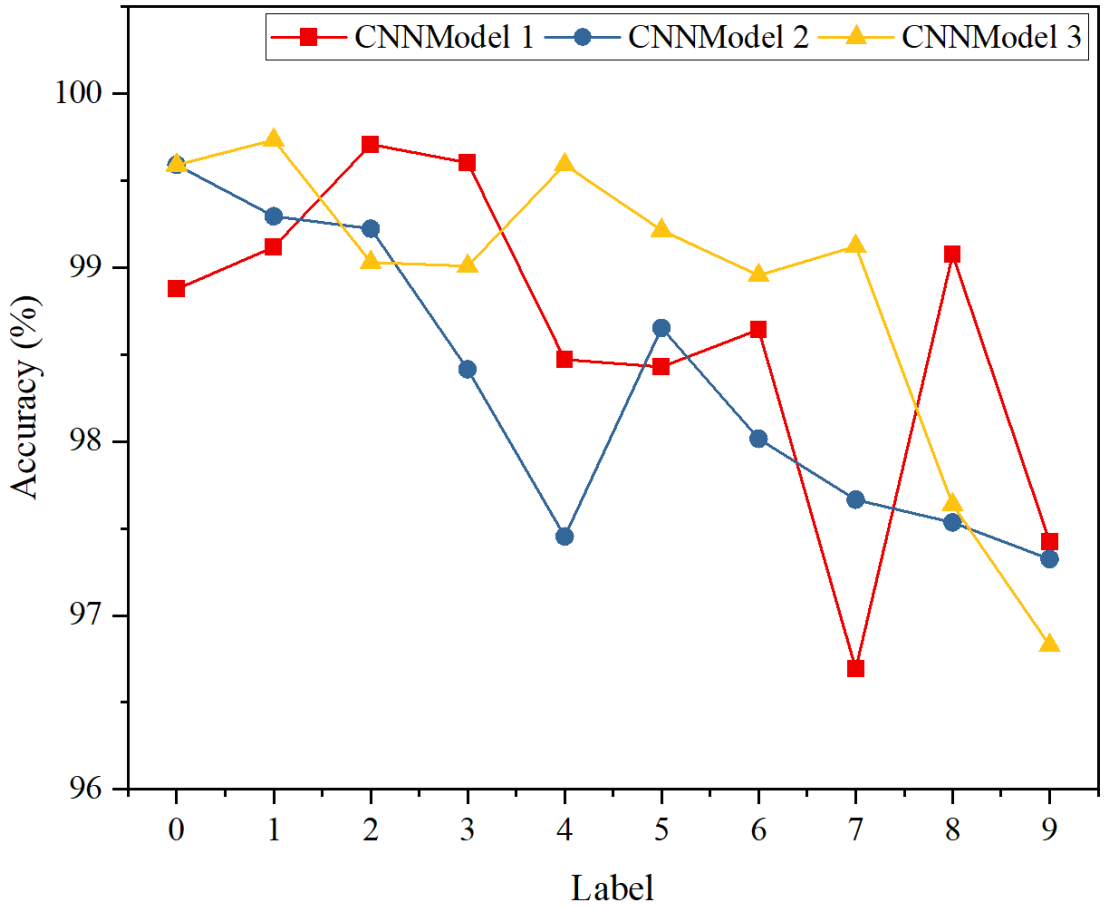

图 15：三组模型在测试集十分类中每一类的准确率

1.4 Discussion（结果分析）

1.4.1 总体结果分析

在上述 3 组神经网络架构下，模型在测试集上的 accuracy 都可以达到 0.98 以上，这说明在训练模型时，有很多种不同的神经网络架构和训练超参数都可以满足要求。在实际训练过程中，需要根据训练效果及时调整网络层数、各层的神经元个数、激活函数以及 epoch 数量等。

观察上面的训练集和测试集上的损失 loss 随 mini-batch 数量的变化趋势图像，可以发现其共同特征是模型在训练集和测试集上的损失 loss 都是相近的（橙色线和蓝色线基本重合），但模型在测试集上的 loss 波动较小，在训练集上的 loss 可能会有较大的突变现象。

模型在训练集和测试集上 loss 的变化趋势都符合以下四步：

①初始阶段大幅下降：在模型刚开始训练时，由于参数的随机初始化，模型可能会以较大的步骤向最优方向优化。此时模型对训练数据完成了初步拟合。

②缓慢下降或波动：当模型不断调整参数时，初始的快速优化会导致模型过拟合训练集，此时模型的性能会出现一段时间的停滞或幅度较小的波动。

③再次大幅下降：卷积神经网络架构中的 Dropout、批量正则化层将使模型更好地泛化。在这个阶段，loss 会再次出现较大幅度的下降，因为是对整体的泛化性能进行优化。

④收敛阶段：最后，loss 的下降速度会减缓，直至最后线条变为几乎水平，最终收敛到一个较小的值，说明模型基本达到收敛。

总体来说，模型在 10 个分类上的准确率 recall 基本都在 0.97 以上，并且相互之间都较为相近，说明模型在均匀分布的测试数据上具有较高的泛化性。

1.4.2 不同组网络架构之间的对比

第一组和第二组的主要区别在于各卷积层的神经元个数不同，第一组为 32，而第二组为 64。实际上两组的收敛速度和准确率都大致相同，说明**本实验中每个卷积层只需要 32 个神经元就可以提取绝大部分的有效特征了**。

第一组和第三组的主要区别在于第三组引入了 3 个 ResNet 模块，而第二组只有 2 个 ResNet 模块。同样地，二者在收敛速度和准确率两个指标上都大致相同，说明**卷积神经网络架构中 ResNet 模块的数量并不会对结果有较大影响，其内部的残差模块的性能可以大幅提升模型的泛化性，解决梯度消失和退化问题**。

1.5 Reference（参考文献）

[1] 华中科技大学计算机学院 2023 年秋季计算机视觉《第五讲 卷积神经网络（上）》
[2] 华中科技大学计算机学院 2023 年秋季计算机视觉《第五讲 卷积神经网络（下）》
[3] 华中科技大学计算机学院 2023 年秋季计算机视觉《第五讲 常见 CNN 网络及深度学习平台 v2.0》
[4] NN-SVG https://github.com/alexlenail/NN-SVG

1.6 Appendix（源代码）

1.6.1 main.py

作用：定义卷积神经网络和 ResNet 模块，训练模型。
代码：

```
import tensorflow as tf
from tensorflow.keras import layers
from tensorflow.keras.callbacks import Callback

# 定义 ResNet 模块
class ResNetBlock(layers.Layer):
    def __init__(self, filters):
        super(ResNetBlock, self).__init__()
        self.conv1 = layers.Conv2D(filters, (1, 1), activation='relu')
        self.bn1 = layers.BatchNormalization()
        self.conv2 = layers.Conv2D(filters, (3, 3), activation='relu',
padding='same')
        self.bn2 = layers.BatchNormalization()
        self.conv3 = layers.Conv2D(filters, (1, 1))
        self.bn3 = layers.BatchNormalization()

    def call(self, inputs):
        x = self.conv1(inputs)
        x = self.bn1(x)
        x = self.conv2(x)
        x = self.bn2(x)
        x = self.conv3(x)
        x = self.bn3(x)
```

```
        x = layers.Add()([inputs, x])
        x = layers.Activation('relu')(x)
        return x

# 定义卷积神经网络结构
class CNNModel(tf.keras.Model):
    def __init__(self):
        super(CNNModel, self).__init__()
        self.conv1 = layers.Conv2D(32, (3, 3), activation='relu',
input_shape=(28, 28, 1))  # 卷积层
        self.bn1 = layers.BatchNormalization()  # 批归一化层
        self.maxpool = layers.MaxPooling2D((2, 2))  # 最大池化层
        self.resnet_block1 = ResNetBlock(32)  # ResNet 模块 1
        self.resnet_block2 = ResNetBlock(32)  # ResNet 模块 2
        self.flatten = layers.Flatten()  # 扁平化层
        self.fc1 = layers.Dense(64, activation='relu')  # 全连接层
        self.dropout = layers.Dropout(0.0001) # Dropout 层
        self.fc2 = layers.Dense(10, activation='softmax') # 全连接层（也
就是输出层）

    def call(self, inputs):
        x = self.conv1(inputs)
        x = self.bn1(x)
        x = self.maxpool(x)
        x = self.resnet_block1(x)
        x = self.resnet_block2(x)
        x = self.flatten(x)
        x = self.fc1(x)
        x = self.dropout(x)
        output = self.fc2(x)
        return output

class LossHistory(Callback):
    def __init__(self, train_data, test_data):
        self.train_data = train_data
        self.test_data = test_data

    def on_batch_end(self, batch, logs=None):
        train_loss = logs.get('loss')
        test_loss = self.model.evaluate(self.test_data[0],
self.test_data[1], verbose=0)
        print("    Mini-Batch:", batch, "- Train Loss:", train_loss, " -
Test Loss:", test_loss[0])
```

```
    def on_epoch_end(self, epoch, logs=None):
        pass

mnist = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train = x_train / 255.0
x_test = x_test / 255.0

x_train = tf.expand_dims(x_train, -1)
x_test = tf.expand_dims(x_test, -1)

model = CNNModel()
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

history = model.fit(x_train, y_train, epochs=10, batch_size=512,
validation_data=(x_test, y_test), callbacks=[LossHistory((x_train,
y_train), (x_test, y_test))])

train_loss, train_acc = model.evaluate(x_train, y_train, verbose=2)
test_loss, test_acc = model.evaluate(x_test, y_test, verbose=2)
print('Train loss:', train_loss)
print('Train accuracy:', train_acc)
print('Test loss:', test_loss)
print('Test accuracy:', test_acc)

import numpy as np
from sklearn.metrics import classification_report

y_pred_probabilities = model.predict(x_test)
y_pred = np.argmax(y_pred_probabilities, axis=-1)

# 使用 classification_report 函数计算每个类别的评估指标
report = classification_report(y_test, y_pred, digits=10,
output_dict=False)
print(report)
```

1.6.2 extract.py

作用：使用正则表达式提取每个 mini-batch 训练后在训练集和测试集上的损失。

代码：

```
import re
import csv
import chardet

input_file = "output.log"
output_file = "Loss.csv"
loss_values = []

with open(input_file, 'rb') as file:
    result = chardet.detect(file.read())
encoding = result['encoding']

with open(input_file, 'r', encoding=encoding) as file:
    for line in file:
        match = re.search(r'Train Loss: (\d+\.\d+)  - Test Loss: 
(\d+\.\d+)', line)
        if match:
            train_loss = float(match.group(1))
            test_loss = float(match.group(2))
            loss_values.append([train_loss, test_loss])

with open(output_file, 'w', newline='') as file:
    writer = csv.writer(file)
    writer.writerow(['Train Loss', 'Test Loss'])
    for values in loss_values:
        writer.writerow(values)
```

1.6.3 draw.py

作用：绘制 Train Loss 和 Test Loss 的变化趋势图。
代码：

```
import csv
import matplotlib.pyplot as plt

input_file = 'Loss.csv'
train_loss = []
test_loss = []

with open(input_file, 'r') as file:
    reader = csv.reader(file)
    next(reader)
    for row in reader:
        train_loss.append(float(row[0]))
        test_loss.append(float(row[1]))
```

```
plt.plot(train_loss, label='Train Loss')
plt.plot(test_loss, label='Test Loss')
plt.xlabel('Mini-Batch')
plt.ylabel('Loss')
plt.title('Train Loss and Test Loss')
plt.legend()

plt.show()
```